Predictable Packet Lossand Proportional Buffer Scaling Mechanism

Ramesh K, Mahesh S. Nayak
Dept .of PG Studies in Computer Science, Karnataka University Dharwd, Karnataka, India
{rvkkud, mnayak.kud}@ gmail.com

Abstract—**Packet losses at IP network are common behavior at the time of congestion. The TCP traffic is explained as in terms of load and capacity. The load should be measured as number of sender actively competes for a bottleneck link and the capacity as the total network buffering available to those senders. Though there are many congestion mechanism already in practice like congestion window, slow start, congestion avoidance, fast transmit but still we see erratic behavior when there is a large traffic. The TCP protocol that controls sources send rates degrades rapidly if the network cannot store at least a few packets per active connection. Thus the amount of router buffer space required for good performance scales with the number of active connections and the bandwidth utilization by each active connections. As in the current practice, the buffer space does not scale in this way and router drops the packet without looking at bandwidth utilization of each connections. The result is global synchronization and phase effect as well as packet from the unlucky sender will be frequently dropped. The simultaneous requirements of low queuing delay and of large buffer memories for large numbers of connections pose a problem. Routers should enforce a dropping policy by proportional to the bandwidth utilization by each active connection. Router will provision the buffering mechanism when processing slows down. This study explains the existing problem with drop-tail and RED routers and proposes the new mechanism to predict the effective bandwidth utilization of the clients depending on their history of utilization and drop the packet in different pattern after analyzing the network bandwidth utilization at each specific interval of time**

Keywords—**Random Early Detection; Drop-Tail; packet dropping probability;**

I. Introduction

Packet buffers in router/switch interfaces constitute a central element of packet networks. The appropriate sizing of these buffers is an important and open research problem. Much of the previous work on buffer sizing modeled the traffic as an exogenous process, i.e., independent of the network state, ignoring the fact that the offered load from TCP flows depends on delays and losses in the network. In TCP-aware work, the objective has often been to maximize the utilization of the link, without considering the resulting loss rate. Also, previous TCP-aware buffer sizing schemes did not distinguish between flows that are bottlenecked at the given link and flows that are bottlenecked elsewhere, or that are limited by their size or advertised window.

The need for buffering is a fundamental "fact of life" for packet switching networks. The TCP traffic is explained as in terms of load and capacity. The load should be measured as number of sender actively competes for a bottleneck link and the capacity as the total network buffering available to those senders. Though there are many congestion mechanism already in practice like congestion window, slow start, congestion avoidance, Fast transmit but still we see erratic behavior when there is a large traffic.

So the router queue lengths should vary with the number of active connections. The TCP protocol that controls sources send rates degrades rapidly if the network cannot store at least a few packets per active connection. Thus the amount of router buffer space required for good performance scales with the number of active connections. If, as in current practice, the buffer space does not scale in this way, the result is highly variable delay on a scale perceptible by users. Router will provision the buffering mechanism when processing slows down. Normally there are two types of algorithm used to determine the buffer size. First, Delay-bandwidth product of memory. This is the minimum amount of buffering that ensures full utilization when number of TCP connection share drop-tail router. With any less than full delay bandwidth product of router memory the TCP window would sum to less than delay bandwidth product of memory. Second is the buffering is only need to absorb transient burst in the traffic. This is true as long as the traffic sources can be persuaded to send at rate that consistently sums to close link bandwidth.

II. The problem

How much packet buffer memory should router ports contain? Previous work suggests two answer. First, one delay-bandwidth product of packet storage. An abstract justification for this is that it allows for the natural delay at which the end system reacts to signals from the network [2, 3]. A TCP specific justification is that this is the minimum amount of buffering that will ensure full utilization when a TCP connection shares a drop-tail router [4]. Another answer is that buffering is only needed to absorb transient bursts in traffic[1].This is true as long as the traffic sources can be persuaded to send at rates that consistently sum to close to the link bandwidth. RED, with its ability to desynchronize TCP window decreases, should be able to achieve this Both of these answers effectively use packet discard as the only mechanism to control load. This thesis suggests using a combination of queuing delay and discard instead.

Much of this thesis deals with how a TCP network should cope with high load and congestion. Load encompasses le gitimate user action tend to place the network under strain and cause it to behave badly. Congestion is bad behavior caused by load. This thesis will argue that an important source or measure of load in TCP network is the number of simultaneous active connections sharing a bottleneck, and that congestion amounts to loss rates high enough to force TCPs into timeout.

The idea that window flow control in general has scaling limits because the window size can't fall below one packet has long been known [5]. Villamizar [4] suggests that this might limit the number of TCP flows a router could support, and that increasing router memory or decreasing packet size could help. Eldridge [6] notes that window flow control cannot control the send rate well on very low-delay networks and advocates use of rate control instead of window control; no specific mechanism is presented. Kung and Chang [7] describe a system that uses a small constant amount of switch buffer space per circuit in order to achieve good scalable performance.

One drawback of drop-tail and RED routers is that they must drop packets in order to signal congestion, a problem particularly acute with limited buffer space and large number of flows. These packets consume bandwidth on the way to the place they are dropped, and they also cause increased incidence of TCP timeouts. The router queue lengths should vary with the number of active connections. The TCP protocol that controls sources send rates degrades rapidly if the network cannot store at least a few packets per active connection. Thus the amount of router buffer space required for good performance scales with the number of active connections and the bandwidth utilization by each active connections. If, as in current practice, the buffer space does not scale in this way and router drops the packet without looking at bandwidth utilization of each connections. The result is global synchronization and phase effect as well as packet from the unlucky sender will be dropped. The simultaneous requirements of low queuing delay and of large buffer memories for large numbers of connections pose a problem. First, routers should have physical memory in proportion to the maximum number of active connections they are likely to encounter. Second, routers should enforce a dropping policy by proportional to the bandwidth utilization by each active connection. This study explains maintaining or varying the router buffer queues depending on number of active connections and dropping policy at router depending on the bandwidth utilization by each active connection. The current practice limits the buffer space to not more than what is required for good utilization. So this study should result some algorithm which helps in varying the buffer size and dropping policy depending on the bandwidth utilization by each active connections.

III. Our solution

A. Idea

When the queue is getting filled up to drop the packet following need to be considered

1. Drop rate (T): When to drop the packet.
2. What to drop: Which packet need to be dropped.
3. How much packets need to be dropped (Ndrop): At each trigger when dropping activity started how much packet need to be dropped.
4. Queue length need to be processed to drop packets (maxp): As entire queue can't be processed to drop the packet which will increase the delay.

Instead of dropping the packet randomly with some probability as stated by RED algorithm, the proposed method will predict the active senders in the network, and then try to drop the packets of the senders depending on their bandwidth utilization which in turn reduce the window size of the active senders.

To achieve this we need to maintain the log history of the processed packets at the head of the queue. These logs are analyzed at particular intervals to find the top recent contributor for the network traffic/congestion.

Flow matrix

Table I: table specifies the top contributer for the traffic at 2ms interval

A1	A2	A3	A4
B1	B2	B3	B4
C1	C2	C3	C4
D1	D2	D3	D4

In the above table A1, D1 etc represent the flow. After analyzing the log which has been updated during the packet processing at the front of the queue, we maintain a table as shown above where the first row represents the flow with highest packet at interval of 2ms. First column represents the recent packet arrivals. Once logs analyzed and the top rows of the table contents are considered as the highest contributor for the network traffic. This will help to decide which packet needs to be dropped when there is queue overflow. The highest packet contributors are predicted to be the top contributor for the further interval as well. So these packets are searched in the queue and dropped depending on the rate at which queue is growing.

The distribution of the flow across the flow matrix is stated as exclusive flow factor as shown below.

$$\text{Exclusive Flow Factor (Eff)} = \frac{\text{Number of Flows}}{\text{Number of unique flow}}$$

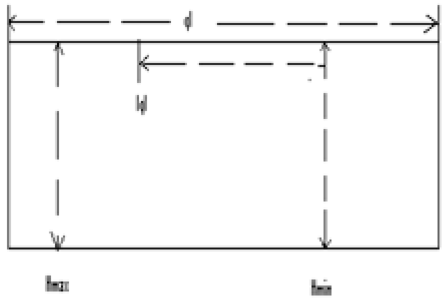


Fig 2: Router Buffer Queue

© 2011 ACEEE
DOI: 01.IJNS.02.04.523

ql is the total queue length of the router. Iql is the instantaneous queue length at a given point of time.

Hmin and Hmax are the minimum and maximum queue length respectively, which administrator can set.

Now once the queue length is exceeded Hmin value, we need to go to end of the queue(Iql) and start processing from there to drop a packet, but how many packets to be processed to drop the packets from the end of the queue is decided by maxp.

$$\text{Maxp} = \frac{\text{Iql} - \text{Hmin}}{\text{Eff}}$$

where Iql is the instantaneous queue length at the time of processing then we need to process maxp number of packet starting from Iql of the queue.

Once the processing started from Iql we will look at the flow matrices to decide which packet need to be dropped. Now once started processing from the end of the queue (Iql) how many packet need to be dropped is calculated by the factor called Ndrop.

$$\text{Ndrop} = \frac{\text{Iql-Hmin}}{\text{(ql-Iql)Eff}}$$

Once all these three parameters are decided (maxp, Ndrop, flow matrix). We need to find the frequency at which we will drop the packet with above considerations. When we look at the rate at which the queue grows will decide on when to drop the packet. So when the queue grows beyond the Hmin at each interval of time (T) queue length is calculated and decided on packet drop,

if (Iql > 0.25(ql-Hmin)+Hmin).

B. Conventional red and mred

In conventional RED algorithm [8], it measures the average queue size to evaluate the degree of network congestion at first. But, due to the quick diversity of network condition, the average queue size of the gateway sometimes can't reflect the network congestion immediately and precisely. For example, if some gateway's queue buffer is filled up with bursty packets and is soon cleared out, the average queue size can't reflect the proper queue buffer status and instantaneous network congestion situation at all. Instead, conventional RED algorithm evaluates the degree of instantaneous network congestion by way of Exponential Weighted Moving Average (EWMA) of the original average queue size as (1):

$$\text{Avg} = (\text{l-w}) \times \text{Avg} + q \times w \quad (1)$$

Here, Avg means the average queue size and q represents the instantaneous queue size. In addition, w is the weighted factor and generally equal to negative power of two. The weighted factor w is a preset constant that determines the sensitivity of RED gateway to the instantaneous queue size. It is usually set to slightly small in order to prevent the average queue size Avg from varying too rapidly and dynamically.

RED algorithm makes use of the average queue size to estimate the network congestion condition and determine the packet drop probability. The packet drop probability depends upon the relationship between the average queue size Avg and two thresholds, maximum threshold Maxth and minimum threshold Minth. As the average queue size Avg is under the minimum threshold Minth, all incoming packets are enqueued sequentially. As the average queue size Avg is over the maximum threshold Maxth, all arriving packet are dropped unconditionally. As the average queue size Avg ranges between the minimum threshold Minth and the maximum threshold Maxth, the nominal packet dropping probability Pb is given by the following equation (2):

$$\text{Pb} = \text{Maxp} \times ((\text{Avg-Minth})/(\text{Maxth- Minth})) \quad (2)$$

Here, Maxp is the preset packet dropping probability. Pb is the nominal packet dropping probability varying from 0 to Maxp linearly and evenly, but RED algorithm throws the packet away by the actual packet dropping probability Pa as (3):

$$\text{Pa} = \text{Pb} / (1\text{-count}\times\text{Pb}) \quad (3)$$

where count means the number of enqueued packets since the last packet is dropped. The actual packet dropping probability Pa is used to make the packet dropping probability raise slowly with the increasing count.

As for MRED algorit hm [9], it is modified from RED algorithm as its name. In addition to the condition of Avg > Maxth, the primary idea of MRED algorithm takes an extra condition of q > Maxth into account to decide if packets are dropped directly. MRED algorithm can provide higher transmission throughput and avoid the sensitivity of RED performance to the parameter setting.

This is because a few parameters of RED algorithm may result in an opposite effect on the queue management mechanism under various network conditions. Against the inaccuracy and inefficiency above, this paper puts concentration on regulating the nominal packet dropping probability progressively by applying more parameters and segmenting the threshold parameters progressively.

C. Proposed RED

```
If (((0.25(Hmax-Hmin)) + Hmin) > Iql > Hmin)
        Identify Packet from the Flow to be dropped
        Process till Hmin
        Drop a Packet
Else If (Hmax>Iql> = ((0.25(Hmax-Hmin)) Hmin)
        Calculate Ndrop
        Calculate maxp
        Identify Drop Flow
        Drop packets
Else If (Iql > Hmax)
        Drop Packets
Else
        Enque the Packets
End If
```

IV. Simulations

A. Environment

Most of the simulations described in this thesis were performed on a uniform configuration designed to highlight buffering and flow count issues. As pictured in Figure 1, this configuration involves N TCP senders converging on router A. Router A must send what data it can across its link to router B, and either buffer or discards the rest. Thus router A is the bottleneck router, and the link from B is the bottleneck link. The intent of this configuration is to capture what happens on heavily loaded links. For example, the link from A to B might be the link from an Internet Service Provider's backbone to a customer. Such links usually run slower than either the backbone or the customer's LAN, and thus act as bottlenecks.

The simulations use the NS 1.4 simulator [10]. Unless otherwise noted, they use the parameters given in Figure 2.

Note that Fig 2 implies that the average round-trip propagation delay is100 milliseconds. Since a 10 megabit link can send 2170 packets per second, the delay-bandwidth product is 217 packets. This means that a single TCP with a 64 kilobyte window can only use about half the link capacity. In addition, each sender's link to router A only runs at 10 times its fair share; this means that if more than 90% of the senders are timing out or otherwise idle, the remaining senders won't be able to use the whole link capacity. In practice this doesn't happen.

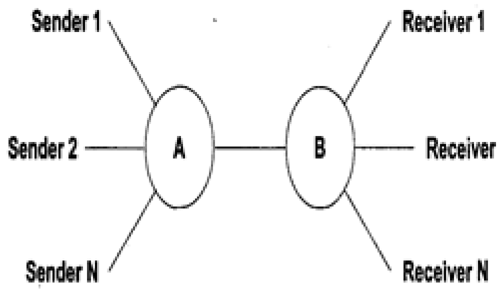
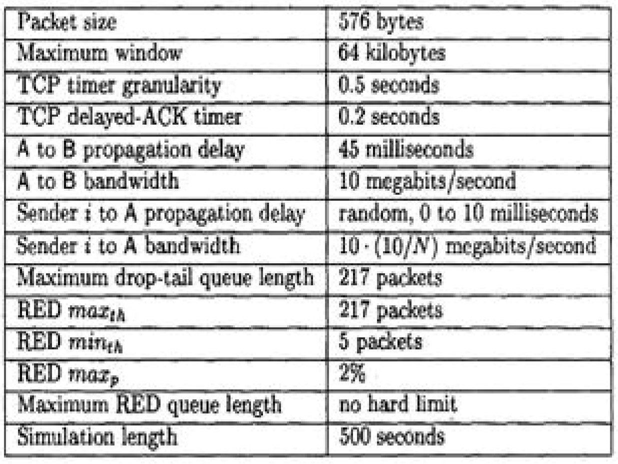


Figure 1: Standard simulation configuration. Each of N TCP senders has its own host and its own link to router A. Router A connects to router B over a bottleneck link



Packet size	576 bytes
Maximum window	64 kilobytes
TCP timer granularity	0.5 seconds
TCP delayed-ACK timer	0.2 seconds
A to B propagation delay	45 milliseconds
A to B bandwidth	10 megabits/second
Sender i to A propagation delay	random, 0 to 10 milliseconds
Sender i to A bandwidth	$10 \cdot (10/N)$ megabits/second
Maximum drop-tail queue length	217 packets
RED max_{th}	217 packets
RED min_{th}	5 packets
RED max_p	2%
Maximum RED queue length	no hard limit
Simulation length	500 seconds

Figure 2: Summary of simulation Parameters

By default, each TCP sender has an unlimited amount of data to send. This means that most of the simulations focus on the steady-state behavior of TCP, rather than its start-up behavior. Some of the simulations use finite length transfers which the relevant chapters will describe as they arise.

The simulations include randomness to avoid deterministic phenomena that would never occur in the real world [11]. The connection start time are randomly spread over the first 10 seconds of the simulation and the senders' access link propagation delays are randomly selected over range of about 10% of the total. Except for the slight randomization, all connections experience about the same round-trip time. This avoids unfairness due to TCP's tendency to use bandwidth in inverse proportion to the round-trip time. Except when noted, no transmission errors are simulated. Thus all packet loss is caused by router A discarding packets in response to congestion. The simulated drop-tail and random drop routers discard packets when the queue already contains a full 217 packets. The RED router drops according to the RED algorithm; there is no hard limit on the amount of buffer memory that RED can use.

The TCP version used in the simulations is Tahoe [12]. The main difference between Tahoe and the later is that Tahoe invokes slow start after a fast retransmit, whereas Reno uses "fast recovery" [13] to continue in congestion avoidance. Section 6.1 will demonstrate that the behavior of Tahoe and Reno are similar for the purposes of this work.

Many of the graphs in this thesis have number of flows on the x axis, and some simulation result (such as loss rate) on the y axis. Each point typically represents the average of that result over a single 500-second simulation. The upper bound on the number of flows presented is typically about 1000. This is not an unreasonable number of flows to expect a 10 megabit link to support, since it results in the per-flow fair share being somewhat slower than a modem.

B. Simulation results

To evaluate the performance of RED, MRED, and PRED algorithms generally and objectively, the experiment tests a variety of network topologies. The experiment performs various n-to-n (n increases from 10 to 100) symmetric network topologies as shown in Fig. 3. The amount of TCP connections increases from 10-to-10 (10x10) to 100-to-100(100x100) for performance comparison between RED, MRED, and PRED algorithms.

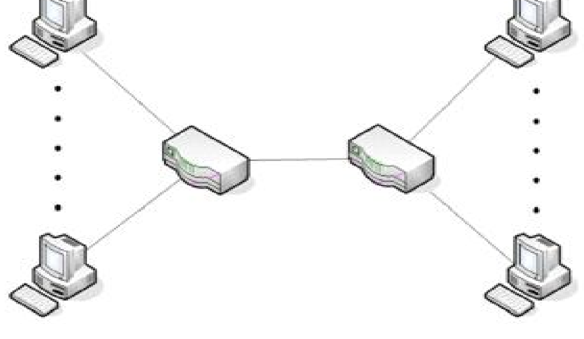


Figure 3. n-to-n network topology for the experiment

TABLE I. AVERAGE THROUGHPUT COMPARISON BETWEEN RED, NRED AND PRED (UNIT OF THROUGHPUT: KBPS)

Topology	10x10	20x20	30x30	40x40	50x50
RED	4782	2325	1321	987	835
MRED	4984	2407	1488	1098	940
PRED	4779	2515	1659	1236	992

Topology	60x60	70x70	80x80	90x90	100x100
RED	673	575	477	415	386
MRED	704	597	500	448	382
PRED	787	675	586	490	464

In the first experiment, this paper performs the average throughput comparison between RED, MRED, and PRED algorithms in each network topology. This paper compares the average throughput between RED, MRED, and PRED algorithms in the network topology of TCP connections varying from 10-to-10 (10x10) to 100-to-100 (100x100). The simulation result is shown in Table I. The total network bandwidth is constant, the more the amount of TCP connections is, the less the average throughput of each TCP connection gets. From Table I, it is obviously verified that proposed PRED algorithm can achieve higher average throughput than conventional RED and MRED algorithms for almost all network topologies.

In the second experiment, this paper performs the normalized throughput comparison between RED, MRED, and PRED algorithms in each network topology. The normalized throughput is defined as the ratio of the number of successfully received packets to the amount of total transmitted packets. Note that the successfully received packets account for both the successfully received non-real time packets and the delay-sensitive packets that satisfy the delay constraint. Fig. 4 shows the normalized throughput comparison between RED, MRED, and PRED algorithms for various network topologies.

From Fig. 4, it is clearly shown that the normalized throughput of conventional RED algorithm is the worst, and that of proposed PRED algorithm is the best. According to Table I and Fig. 4, we can see that the proposed PRED algorithm can realize better network bandwidth utilization and queue management efficiency under various network conditions.

In the third experiment, this paper performs the end-to-end average delay comparison between the transmitter and receiver. Fig. 5 shows the simulation result about the end-to-end average delay comparison between RED, MRED, and PRED algorithms for various network topologies. From Fig.5, we can see that the proposed PRED algorithm also can decrease the end-to-end average delay against conventional RED and MRED algorithms.

In the last experiment, Fig. 6 illustrates the instantaneous queue size in the queue buffer versus the operating time for RED, MRED, and PRED gateways.

The maximum queue size in the queue buffer in the experiment is set to 50 packets. From Fig. 6, we can see the proposed PRED algorithm regulates the queue size in the queue buffer of the gateway progressively and smoothly.

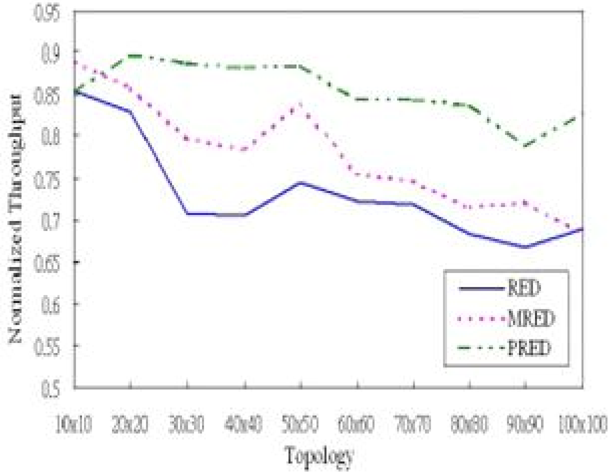


Figure.4 Normalized throughput comparison between RED, MRED and PRED for various network topologies

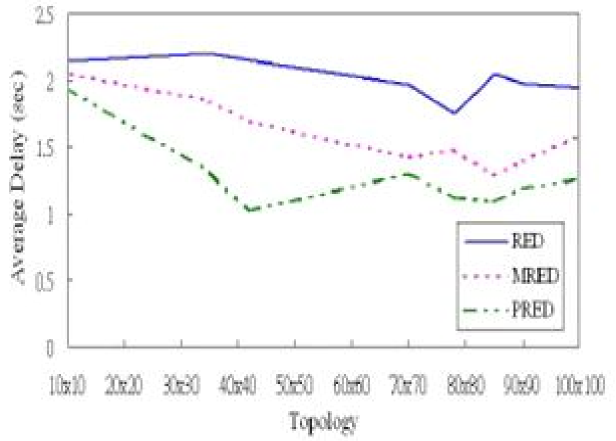


Figure. 5 End-to-end average delay comparisons between RED, MRED and PRED for various network topologies

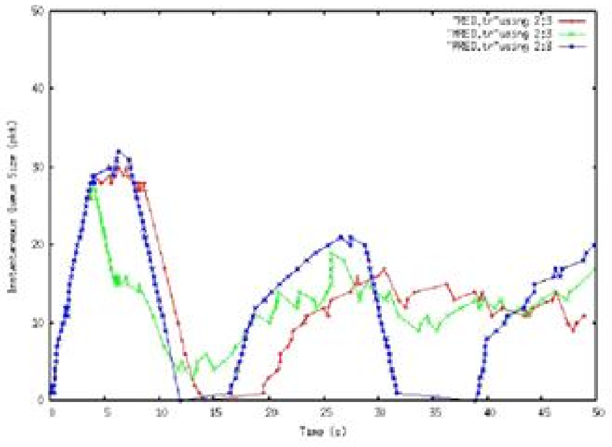


Figure 6 Instantaneous queue size situations of RED, MRED and PRED gateways

V. CONCLUSION

This study explains maintaining or varying the router buffer queues depending on number of active connections and dropping policy at router depending on the bandwidth utilization by each active connection. The current practice limits the buffer space to not more than what is required for good utilization. So this study should result some algorithm which helps in varying the buffer size and dropping policy depending on the bandwidth utilization by each active connections.

From the viewpoint of the instantaneous queue size, the PRED algorithm is proposed to adjust the maximum threshold dynamically to adapt to various network conditions, and regulates the packet dropping probability progressively by comparing the instantaneous queue size with the progressive maximum queue threshold parameters.

According to all aspects of experimental results, the proposed PRED algorithm is shown to realize better network bandwidth utilization and queue management efficiency under various network conditions. Also, the end-to-end average delay can further be reduced by the proposed PRED algorithm. More importantly, the proposed progressive adjustment method can also be applied to other RED-based algorithms.

References

1. B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenkar, J. Wroc lawski, L. Zhang, “ Recommendations On Queue Management and Congestion Avoidance in the Internet”, RFC 2309, April.1998.
2. H. T. Kung and Alan Chapman. The fcvc (flow controlled virtual channels) proposal for an atm networks. In Proceedings of the International Conference on Network Peotocols, 1993.
3. H. T. Kung, Trevor Blackwell, and Alan Chapman. Credit based flow controlfor an atm network: Credit update protocol, adaptive credit allocation, and statistical multiplexing. In Proceedings of ACM SIGCOMM, 1994.
4. Curtis Villamizar and Cheng Song. High performance tcp in ansnet. Computer Communication Reviews, 24(5), October 1994.
5. Dimitri Bertsekas and Robert Gallager. Data Networks. Prentice Hall, Englewood Cloffs, New Jersey, 1987.
6.Charles Eldrige. Rate controls in standard transport layer protocols. Computer Communication Reviews, 22(3), July 1992.
7. H. T. Kung and Koling Chang. Receiver-oriented adaptive buffer allocation in credit based flow control for an atm networks. In Proceedings of IEEE Infocom, 1995.
8. Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. IEEE/ACM Transactions on Networking, August 1993.
9. Gang Feng and A. K. Agarwal and A. Jayaraman and Chee Kheong Siew, “Modified RED gateways under bursty traffic,” IEEE Communications Letters, vol. 8, pp. 323-325, May 2004.
10. Steve McCanne and Sally Floyd. Ns (Network Simulator). http://www.nrg.ee.lbl.gov/ns/, June 1998.
11. Trevor Blackwell. Applications of Randomness in System Performance Measurement. PhD thesis, Harward University, 1998.
12. W. Richard Stevens. Rfc2001: Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. Technical Report, Internet Assigned Numbers Authority, John Postel, USC/ ISI, 4676 Admiralty Way, Marina del Rey, DA 90292, 1997. http:/ /info.internet.isiedu/in-notes/rfc/files/rfc2001.txt.
13. W. Richard Stevens. TCP/IP Illustrated Volume 1: The Protocols. Addison-Wesley, 1994.